

프로그래밍 실습 #10

채진석 교수님
201702797 채승민

1. 다음 암호화 알고리즘을 통해 암호화된 메시지를 복호화하는 프로그램을 작성하라.

(1) 카이사르 암호화 알고리즘

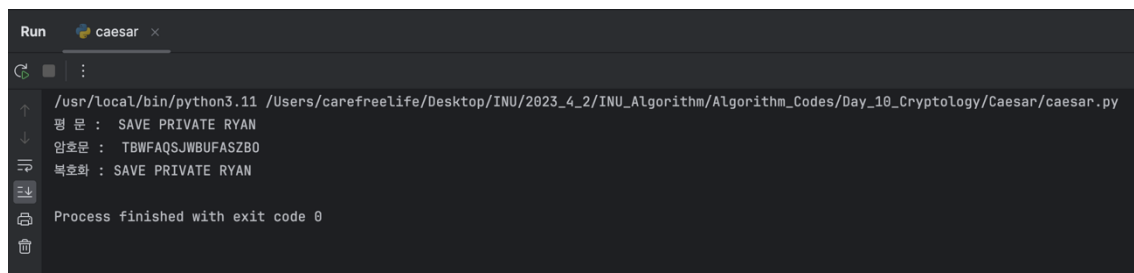
[소스코드]

```
def encipher(plain, k):
    n = len(plain)
    cipher = ''
    for i in range(n):
        a = ord(plain[i])
        if a == 32:
            a = 64
        t = a + k
        if t > 90:
            t -= 27
        if t == 64:
            t = 32
        cipher += chr(t)
    return cipher

def decipher(ct, k):
    n = len(ct)
    dec = ''
    for i in range(n):
        a = ord(ct[i])
        if a == 64:
            a = 32
        t = a - k
        if t > 90:
            t -= 27
        if t == 64:
            t = 32
        dec += chr(t)
    return dec

if __name__ == "__main__":
    plainText = 'SAVE PRIVATE RYAN'
    K = 1
    print('평 문 : ', plainText)
    cipherText = encipher(plainText, K)
    print('암호문 : ', cipherText)
    decText = decipher(cipherText, K)
    print(f'복호화 : {decText}')
```

[실행 결과]



```
Run caesar x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_10_Cryptography/Caesar/caesar.py
평 문 : SAVE PRIVATE RYAN
암호문 : TBWFAQSJWBUFASZBO
복호화 : SAVE PRIVATE RYAN
Process finished with exit code 0
```

(2) 문자변환표 사용 암호화 알고리즘

[소스코드]

```
def encipher(plain, k):
    n = len(plain)
    cipher = ''
    for i in range(n):
        a = ord(plain[i])
        if a == 32:
            a = 0
        else:
            a -= 64
        cipher += k[a]
    return cipher

def decipher(dec, k):
    n = len(dec)
    cipher = ''
    for i in range(n):
        idx = k.index(dec[i])
        if idx == 0:
            idx = 32
        else:
            idx += 64
        cipher += chr(idx)
    return cipher

if __name__ == "__main__":
    plainText = 'SAVE PRIVATE RYAN'
    K = 'QHCBEJKARWSTUVD IOPXZFGMLNMY'
    print('평 문 : ', plainText)
    cipherText = encipher(plainText, K)
    print('암호문 : ', cipherText)
    decText = decipher(cipherText, K)
    print(f"복호문 : {decText}")
```

[실행결과]



```
Run convertTable x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_10_Cryptography/ConvertTable/convertTable.py
평 문 : SAVE PRIVATE RYAN
암호문 : XHGJQIPWGHZJQPNHD
복호문 : SAVE PRIVATE RYAN
Process finished with exit code 0
```

(3) 비즈네르 암호화 알고리즘

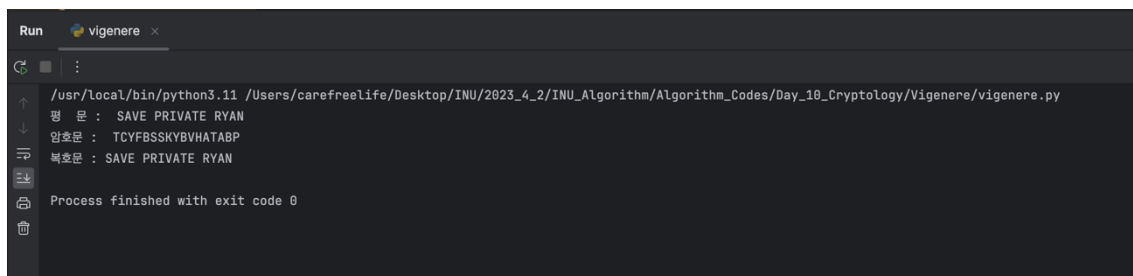
[소스코드]

```
def encipher(p, k):
    c = ''
    n = len(k)
    for i in range(len(p)):
        a = ord(p[i])
        if a == 32:
            a = 64
        b = ord(k[i % n]) - 64
        t = a + b
        if t > 90:
            t -= 27
        if t == 64:
            t = 32
        c += chr(t)
    return c

def decipher(c, k):
    p = ''
    n = len(k)
    for i in range(len(c)):
        a = ord(c[i])
        if a == 32:
            a = 64
        b = ord(k[i % n]) - 64
        t = a - b
        if t < 65:
            t += 27
        if t == 91:
            t = 32
        p += chr(t)
    return p

if __name__ == "__main__":
    plainText = 'SAVE PRIVATE RYAN'
    K = 'ABC'
    print('평 문 : ', plainText)
    cipherText = encipher(plainText, K)
    print('암호문 : ', cipherText)
    decText = decipher(cipherText, K)
    print(f"복호문 : {decText}")
```

[실행 결과]



```
Run vigenere x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_10_Cryptography/Vigenere/vigenere.py
평 문 : SAVE PRIVATE RYAN
암호문 : TCYFBSSKYBVHATABP
복호문 : SAVE PRIVATE RYAN
Process finished with exit code 0
```

(4) RSA 암호화 알고리즘

[소스코드]

```
import re

def encipher(plain, n, p):
    cipher = ''
    i = 0
    while i < len(plain):
        m = ''
        for j in range(4):
            m += plain[i+j]
        i += 4
        a = int(m)
        t = a
        for k in range(p):
            b = t % n
            t = a * b
        if b < 10:
            cipher += '000' + str(b)
        elif b < 100:
            cipher += '00' + str(b)
        elif b < 1000:
            cipher += '0' + str(b)
        else:
            cipher += str(b)
    return cipher

def decipher(enc, s, n):
    recover = ''
    loop = len(enc) // len(str(n))
    while loop:
        tg = int(enc[:4])
        enc = enc[4:]
        # print(f"tg={tg}, enc={enc}")

        concatstr = str(pow(tg, s) % n)
        match len(concatstr):
            case 1:
                concatstr = '000' + concatstr
            case 2:
                concatstr = '00' + concatstr
            case 3:
                concatstr = '0' + concatstr

        recover += concatstr
        loop -= 1
    return recover

def encode(plain):
    n = len(plain)
    m = ''
    for i in range(n):
        a = ord(plain[i])
        if a == 32:
            a = 64
```

```

        a -= 64
        if a == 0:
            m += '00'
        elif a < 10:
            m += '0' + str(a)
        else:
            m += str(a)
    return m

def decode(recov):
    n = len(recov)
    m = ''
    for i in range(0, n, 2):
        temp_d = int(recov[i:i+2]) + 64
        if temp_d == 64:
            temp_d = 32
        m += chr(temp_d)
    return m

if __name__ == "__main__":
    plainText = 'SAVE PRIVATE RYAN '
    N = 3713
    # 비밀 키
    S = 97
    P = 37
    plainMessage = encode(plainText)
    print('평 문 : ', plainMessage)
    cipherMessage = encipher(plainMessage, N, P)
    print('암호문 : ', cipherMessage)
    recov = decipher(cipherMessage, S, N)
    print(f"복호문 : {recov}")
    origin = decode(recov)
    print(f"복호화된 Text : {origin}")

```

[실행결과]

```

Run  rsa x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_10_Cryptography/RSA/rsa.py
평 문 :  190122050016180922012005001825011400
암호문 :  033514721447306015481608309106541414
복호문 :  190122050016180922012005001825011400
복호화된 Text :  SAVE PRIVATE RYAN
Process finished with exit code 0

```

<코딩 테스트 연습 #4>

□ DNA 비밀번호

[문제]

평소에 문자열을 가지고 노는 것을 좋아하는 민호는 DNA 문자열을 알게 되었다. DNA 문자열은 모든 문자열에 등장하는 문자가 {'A', 'C', 'G', 'T'} 인 문자열을 말한다. 예를 들어 "ACKA"는 DNA 문자열이 아니지만 "ACCA"는 DNA 문자열이다. 이런 신비한 문자열에 완전히 매료된 민호는 임의의 DNA 문자열을 만들고 만들어진 DNA 문자열의 부분문자열을 비밀번호로 사용하기로 마음먹었다.

하지만 민호는 이러한 방법에는 큰 문제가 있다는 것을 발견했다. 임의의 DNA 문자열의 부분문자열을 뽑았을 때 "AAAA"와 같이 보안에 취약한 비밀번호가 만들어 질 수 있기 때문이다. 그래서 민호는 부분문자열에서 등장하는 문자의 개수가 특정 개수 이상이어야 비밀번호로 사용할 수 있다는 규칙을 만들었다.

임의의 DNA 문자열이 "AAACCTGCCAA" 이고 민호가 뽑은 부분문자열의 길이를 4 라고 하자. 그리고 부분문자열에 'A' 는 1 개 이상, 'C'는 1 개 이상, 'G'는 1 개 이상, 'T'는 0 개 이상이 등장해야 비밀번호로 사용할 수 있다고 하자. 이때 "ACCT" 는 'G' 가 1 개 이상 등장해야 한다는 조건을 만족하지 못해 비밀번호로 사용하지 못한다. 하지만 "GCCA" 은 모든 조건을 만족하기 때문에 비밀번호로 사용할 수 있다.

민호가 만든 임의의 DNA 문자열과 비밀번호로 사용할 부분문자열의 길이, 그리고 {'A', 'C', 'G', 'T'} 가 각각 몇번 이상 등장해야 비밀번호로 사용할 수 있는지 순서대로 주어졌을 때 민호가 만들 수 있는 비밀번호의 종류의 수를 구하는 프로그램을 작성하자. 단 부분문자열이 등장하는 위치가 다르다면 부분문자열이 같다고 하더라도 다른 문자열로 취급한다.

[입력]

- 첫 번째 줄에 민호가 임의로 만든 DNA 문자열 길이 $|S|$ 와 비밀번호로 사용할 부분문자열의 길이 $|P|$ 가 주어진다. ($1 \leq |P| \leq |S| \leq 1,000,000$)
- 두 번째 줄에는 민호가 임의로 만든 DNA 문자열이 주어진다.
- 세 번째 줄에는 부분문자열에 포함되어야 할 {'A', 'C', 'G', 'T'}의 최소 개수가 공백을 구분으로 주어진다. 각각의 수는 $|S|$ 보다 작거나 같은 음이 아닌 정수이며 총 합은 $|S|$ 보다 작거나 같음이 보장된다.

[출력]

첫 번째 줄에 민호가 만들 수 있는 비밀번호의 종류의 수를 출력하라.

[소스코드]

```
import random
import re

dna = ['A', 'C', 'G', 'T']

def make_rand_dna():
    while True:
        s_length, _p = (input("임의의 DNA 문자열 s의 길이와 비밀 번호로 사용할 부분 문자열의 길이를 입력하세요 (종료: 999) : ")
                        .split(' ', 2))
        s_length = int(s_length)
        _p = int(_p)
        if _p == 999:
            exit(0)
        if 1 <= _p <= s_length <= 1000000:
            break
        print("1 <= p <= len(s) <= 1000000 조건을 충족하는 p를 입력하세요.")
        # rand_dna_length = random.randint(1, s_length)
        rand_dna = ''
        for _ in range(s_length):
            rand_dna_idx = random.randint(0, 3)
            rand_dna += dna[rand_dna_idx]

        print(f"임의의 DNA 문자열 s = {rand_dna}")
        return rand_dna, _p

def input_num(s):
    while True:
        _count = input("A, C, G, T의 등장 횟수를 공백으로 구분하여
```



```

입력하세요: ").split(' ', 4)
    # 정수 리스트 변환
    _count = list(map(int, _count))
    sum = 0
    for i in range(len(_count)):
        if 0 <= _count[i] <= len(s):
            sum += _count[i]
        else:
            print(f"등장 횟수가 잘못 설정 되었습니다. 다시 입력하세요: ")
            break
    if i == len(_count) - 1 and sum <= len(s):
        return _count

# 전체 문자열 s를 p의 길이와 같은 크기의 sliding window로 찾으며 각 원소의
개수를 비교.
def search(s, p, c):
    usable_case = 0
    for i in range(len(s)):
        elem_count = [0 for i in range(len(c))]
        for window in s[i:i+p]:
            match window:
                case 'A':
                    elem_count[0] += 1
                case 'C':
                    elem_count[1] += 1
                case 'G':
                    elem_count[2] += 1
                case 'T':
                    elem_count[3] += 1

        if elem_count == c:
            print(f"가능한 비밀번호: {s[i:i+p]} / Text의 {i} ~ {i+p}
위치")
            usable_case += 1
    return usable_case

def test_case(test_num):
    if test_num == 1:
        return "CCTGGATTG", 8
    elif test_num == 2:
        return "GATA", 2

if __name__ == "__main__":
    # 자동 문자열 생성
    S, P = make_rand_dna()

    # 테스트 문자열 생성
    # S, P = test_case(2)

    count = input_num(S)
    result = search(S, P, count)
    print(result)

```

[실행결과]

```
Run codingtest4 x
/usr/local/bin/python3.11 /Users/carefreelife/Desktop/INU/2023_4_2/INU_Algorithm/Algorithm_Codes/Day_10_Cryptology/CodingTest4/codingtest4.py
임의의 DNA 문자열 S의 길이와 비밀 번호로 사용할 부분 문자열의 길이를 입력하세요 (종료: 999) : 30 5
임의의 DNA 문자열 S = CGTGCCGTAGGCCACTAAAGACATCTGCAA
A, C, G, T의 등장 횟수를 공백으로 구분하여 입력하세요: 2 1 1 1
가능한 비밀번호: GACAT / Text의 19 ~ 24 위치
가능한 비밀번호: TGCAA / Text의 25 ~ 30 위치
2

Process finished with exit code 0
```

201702797 채승민
한 한기동안 감사했습니다.